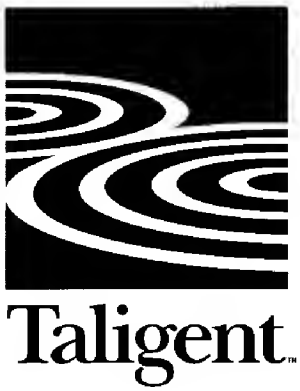
Leveraging
OBJECT-ORIENTED
FRAMEWORKS



LEVERAGING OBJECT-ORIENTED FRAMEWORKS

A TECHNOLOGY PRIMER
FROM TALIGENT

Table of Contents

Leveraging Object-Oriented Frameworks

Introduction

Over the last two decades, software development has changed significantly. Its evolution has been motivated largely by the quest to help developers produce software faster and to deliver more value to end-users. Despite gains, the industry still faces long development cycles which produce software that usually doesn't address business problems adequately – pointing to the limitations of traditional programming and today's system software environments. These limitations have moved the software industry to embrace object-oriented technology (OOT) because of its potential to significantly increase developer productivity and encourage innovation.

Taligent believes that OOT indeed has the potential to dramatically improve the software development process. However, we believe the focus should not only be on OOT, but on how this technology is delivered in order to fully realize the benefits that it has to offer. In our view, object-oriented frameworks – extensible sets of object-oriented classes that are integrated to execute well-defined sets of computing behavior – provide the necessary foundation for fully exploiting the promises of OOT.

Which is why Taligent is building a new system software platform and integrated development environment based entirely on OOT and object-oriented frameworks. The Taligent environment will:

- Empower developers to fully leverage OOT with frameworks that span the entire system. The pervasive use of frameworks in the Taligent system will deliver rich built-in functionality at all levels and provide more computing value than when this functionality is added on as an option.
- Provide well-defined mechanisms that allow software and hardware developers to reuse our frameworks to extend and leverage this functionality for increased productivity and integration.
- Provide an integrated development environment designed for object-oriented programming (OOP) that includes a wide variety of development tools all designed for rapid application development and customization.

This primer describes object-oriented frameworks in the context of other development approaches and how Taligent has utilized them to realize the benefits of object technology. After you read this primer, we hope you have a better understanding of frameworks and Taligent's design approach, how you can benefit from frameworks, and how you can prepare for frameworks and the Taligent system. Within this primer, when we mention frameworks, unless we specify otherwise, we mean object-oriented frameworks, as opposed to frameworks based on other technologies.

This primer is intended mainly for software developers and designers in commercial and corporate software development organizations. However, hardware designers, strategic technology planners, and technical managers will also find this primer useful. We assume the reader has at least some knowledge of OOT and may have little to no familiarity with frameworks. Refer to the "Additional Reading" section in the back for more information on object technology and frameworks.

This particular primer discusses frameworks, which are certainly a cornerstone of the Taligent system. However, future technology primers and white papers will explore other aspects of our system so as to highlight a variety of benefits and points of differentiation which will be important to corporate and commercial software developers, system integrators, hardware OEMs, and other future Taligent partners ad customers.

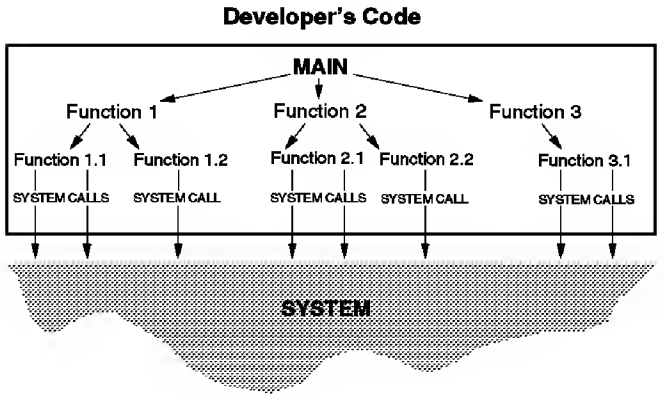
We begin this primer by comparing various software development approaches in order to identify the problems that object technology can solve.

Software Development Approaches

Procedural Programming

In a procedural environment, the developer writes an application by making a series of calls to library routines provided by the system (as well as to routines written by the developer) as shown below in **Figure 1**. The developer's code sits on top of the system code. The develop's code can access all of the system's services, but the system knows nothing about the developer's code. The developer is responsible for providing the overall behavior and flow of control of the application, with the system providing the functionality.

Figure 1



The procedural approach (and its more disciplined offspring, structured programming) has produced major improvements in the quality of software over the last twenty years, but its limitations are painfully apparent due to the:

- **Difficulty in extending and specializing functionality:** Because procedural systems do not provide flexible interfaces, developers cannot selectively change or extend the structure or behavior. It is usually "all or nothing." For example, if a developer has a procedural library that provides basic text-editing capabilities, adding new functionality such as tabs or subscripts in a procedural language, is usually impossible without a great deal of additional coding.
- **Difficulty in factoring out common functionality for reuse:** It is difficult to consolidate common system functions that easily allows reuse by other applications. To consider the impact on corporate developers, imagine a developer working in a large international bank that needs to create pricing and valuation tools for financial instruments. All the tools have common functionality whether it involves interest rate swaps, stock options, or foreign currency. Since it is difficult to factor out the common pieces in a procedural system, this functionality has to be duplicated every time a new instrument is introduced, resulting in a longer time to market and a loss of competitive advantage for the bank.
- **Barriers to interoperability:** Even when extensions and modifications are made in a procedural application, it is hard to ensure that the changes will interoperate correctly with other systems that depend on the modifications. The result is that the solutions from one developer might not have anything in common with any other developer's solutions. So, instead of a small team of experts solving a particular problem once, there are numerous teams repeatedly and unsatisfactorily addressing the same problem.
- **Maintenance overhead:** Since there is minimal reuse of code in a procedural system, maintenance requirements increase due to the greater amount of coding involved and the subsequent increased potential for introducing new bugs.

This lack of extensibility, factorability, interoperability, and maintainability adds up to lower code and design reuse. The result is that developer productivity is severely hindered since more time and resources are spent writing code instead of solving new problems.

Object-Oriented Programming

Although the principles of procedural programming have improved the clarity and reliability of programs, large-scale programming still remains a challenge. Object-oriented programming (OOP) brings a new approach to that challenge.

Unlike procedural programming, which emphasizes algorithms and procedures, OOP emphasizes the binding of data structures with the methods to operate on the data. The idea is to design object classes that correspond to the essential features of a problem. Rather than trying to fit a problem to the procedural approach of a computer language, OOP allows the programmer to use the language to effectively model and solve real-world problems. A drawing program, for example, might define classes that represent rectangles, polygons, and circles. The class definitions would include common functionality that is the same for each class, such as move and rotate. Then a developer would proceed to design a program by deriving subclasses and overriding existing methods or implementing new methods within each class.

This development approach allows developers to break problems into small, manageable modules of code, where the principle of encapsulation insulates developers from having to know the implementation details. Due to the principle of inheritance, developers can subclass to derive new classes from existing ones and be provided with the "hooks" to add extensions.

In addition, polymorphism gives the developer flexibility to create multiple definitions for functions. This allows classes to be more general and hence more reusable. It also allows new components and functions to be added easily and without disturbing the existing system. Implementing OOP makes it possible to design software that is more extensible, reusable, and maintainable.

By helping developers design and produce code more productively, the advantages of OOP have proven to be a significant revolution over traditional programming techniques. However, even though the programming job is made easier, since the developer works at a higher level of abstraction with objects and class libraries,

Object-Oriented Technology Concepts

Objects are software entities that combine data structures and operations on the data. Together, these enable groups of objects to model real-world entities based on their characteristics (represented by data elements) and their behavior (represented by data manipulation operations). In this way, objects can model concrete things such as people and data entry forms and abstractions such as numbers or geometrical concepts.

The benefits of object technology arise out of three basic principles: encapsulation, inheritance, and polymorphism.

Encapsulation: Objects hide or encapsulate the internal structure of their data and the algorithms by which their functions work. Instead of exposing these implementation details, objects present interfaces that represent their abstractions cleanly with no extraneous information.

Inheritance: Inheritance is the main contributor to the increased programming productivity from OOT. Inheritance, which allows developers to reuse pre-existing code, has to do with deriving existing behaviors from classes. A class (in C++) is the specification that describes the data model and the operations that can be performed on that data, and an object is an example of a particular data structure constructed according to that plan. For instance, a class could describe the general properties of a corporate executive (name, title, unusual skills), while an object would represent a specific executive (John Doe, vice president, speaks seven languages). Through inheritance, developers derive subclasses to meet their particular needs.

Polymorphism: Polymorphism takes encapsulation a step further: many shapes, one interface. A software component can make a request of another component without knowing exactly what that component is. The component that receives the request interprets it, and determines according to its variables and data how to execute the request.

the developer still has to "put the pieces together." Simply changing from procedural techniques to OOP does not fix the problem which is that developers still are responsible for providing infrastructure and are not provided with a clean mechanism for extending functionality. Even with OOP, developers write a lot of code since they are still responsible for providing the flow of control of the application. Frameworks, as we will see next, carry the OOP paradigm further by providing infrastructure and flexibility for deploying OOT.

Framework-Oriented Programming

Taligent defines framework-oriented programming as the exploitation of object-oriented frameworks to maximize the benefits of OOT. Although frameworks are not new to the software industry, there is a great deal of discussion about them in object technology circles. What exactly are frameworks? A widely accepted definition comes from Ralph E. Johnson of the University of Illinois:

> *"A framework is a set of classes that embodies an abstract design for solutions to a family of related problems."*[1]

Another way of looking at frameworks is as a prefabricated structure, or template, of a working program. For example, an application framework provides the support and "default" behavior for drawing windows, scrollbars, and menus.

Taligent believes that frameworks, which are central to its new operating environment, are the most important advancement in OOT. Because frameworks provide infrastructure and flexible interfaces, they avoid the problems and overhead that traditional programming impose on developers. With well-designed frameworks, it is mush easier to add extensions, to factor out common functionality, to enable interoperability, and to improve software maintenance and reliability.

1. "Designing Reusable Classes", The Journal of Object-Oriented Programming, Vol. 1, No. 2, 1988, pp 22-35

The way that frameworks, in general, achieve these benefits over other development approaches are based on two fundamental principles:

- **Frameworks provide infrastructure and design:** Frameworks are not simply collections of classes. Rather, frameworks come with rich functionality and strong "wired-in" interconnections between the object classes that provide an infrastructure for the developer. It is these interconnections that provide the architectural model and design for developers and frees them to apply their expertise on the problem domain. By providing an infrastructure, the framework dramatically decreases the amount of standard code that the developer has to program, test, and debug. The developer writes only the code that extends or specifies the framework behavior to suit the program's requirements. We represent this code visually as a "puzzle piece" since this is the creative and undefined part the developer provides (**see Figure 2**).

Taligent believes that frameworks, which are central to its new operating environment, are the most important advancement in OOT. Because frameworks provide infrastructure and flexible interfaces, they avoid the problems and overhead that traditional programming impose on developers.

- **The framework calls you, you don't call the framework:** Framework-oriented programming requires a new way of thinking. In procedural systems, the developer's own program provides all of the structure and flow of execution and makes calls to function libraries as necessary (**see Figure 2**). However, in framework-oriented programming, the roles are turned around. The role of the framework is to provide the flow of control, while the developer's code waits for the call from the framework. This is a significant benefit since developers do not have to be concerned with the details, but can focus their attention on their particular problem domain.

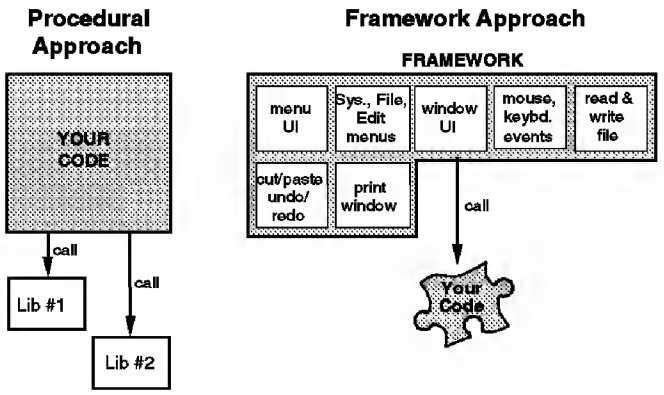


Figure 2

However, this flip-flop in control can be a significant change for developers experienced only in procedural programming. The developer must learn to think in terms of the responsibilities of the objects – what are the objects required to do – and let the framework determine when the objects should do it. Once the investment has been made to understand frameworks, developers will begin to realize the enormous advantages that framework-oriented programming can deliver over other development approaches.

Applying Frameworks – An Overview

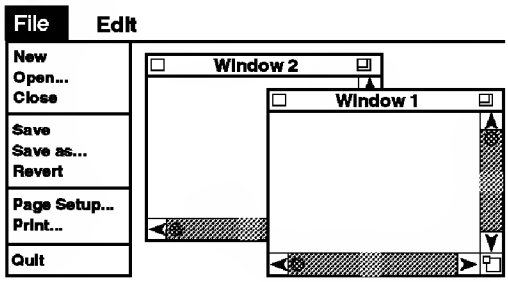
There are many types of frameworks (not all object-oriented) on the market for solving various types of problems. The types of frameworks range from application frameworks that assist in developing the user interface, to lower-level frameworks that provide basic system software services such as communication, printing, and file systems support. Within this range there are also domain-specific frameworks that address problems in particular areas. There are quite a number of commercially available application frameworks. MacApp® (Apple Computer), and OWL™ (Borland International) are examples of application frameworks. There are far fewer system-level frameworks, but examples include the virtual memory and process scheduling frameworks that are part of the CHOICES operating system (University of Illinois).

To illustrate the principles and benefits of frameworks, let's look at how a typical application framework is used. Later, we will show how Taligent is applying the same framework principles and concepts not only at the application level but throughout all levels of its system software environment, thus building the system "from the bottom up" with frameworks.

Applying Application-Level Frameworks

Out of the box, a typical application framework provides the developer with the basic functionality of an application, such as File and Edit menus, windows and printing. **Figure 3** shows a very simple application before the developer starts adding any major customizations.

Figure 3



Initially, the program does not provide any real user functionality – it is still an empty program template. It's up to the developer to fill in the functionality that is unique to the application, such as the specific commands in the new menus, the text and graphics that go in the windows, the routines to write data to a file on disk, and the actions to be carried out when the user clicks the mouse.

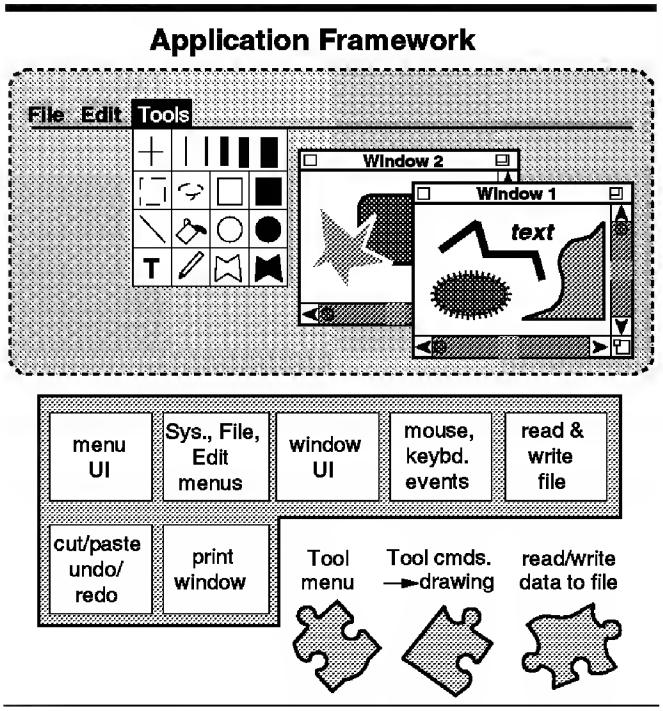
For example, the developer could add a "Tools" menu that brings up a tool palette and allows the user to select and use various tools.

To do this, the developer creates or modifies classes that:

- *Add a new Tool menu.*
- *Display the Tool palette.*
- *Add the behaviors represented by the tools in the Tool palette.*
- *Change the standard arrow cursor so that when the mouse is clicked on the tool palette, the cursor changes to a cross ("+"), I-Beam, paintbrush, or eraser depending on what tool is selected.*

You now have the modest beginnings of a working drawing application as shown in the upper part of **Figure 4**. The developer would continue to take advantage of the application framework's object-oriented characteristics to extend the application further.

Figure 4



In the lower part of Figure 4, we show how in effect, you could create an application by customizing the default behavior of the application framework. This might involve subclassing certain framework classes and overriding specific methods, creating instances of certain other classes, or providing entirely new classes. Overridden methods will be called by the framework when the framework designers think they should be called, thus changing the default behavior. The puzzle pieces below the framework represent the code that developers write to add their specific functionality. The framework calls the behaviors the developer wrote to perform the intended action.

Applying System-Level Frameworks

In the same way that an application framework provides the developer with prefabricated functionality and extensibility at higher levels of the system, system-level frameworks leverage the same concepts and provide similar benefits at lower levels of the system. One advantage of applying frameworks at lower levels of the system is that this approach allows the system to be extended to add new kinds of hardware devices. In the longer term, a more important advantage appears. As hardware technology evolves, it is not possible to predict where the system can be usefully extended. Frameworks provide a mechanism to extend the system virtually everywhere, not just where the system developer thought the system might be extended.

As hardware technology evolves, it is not possible to predict where the system can be usefully extended. Frameworks provide a mechanism to extend the system virtually everywhere, not just where the system developer thought the system might be extended.

Applying frameworks at the system level is a strategy that has not been fully exploited to the extent that Taligent will with our object-oriented system software platform which we will discuss in more detail later. To illustrate the benefits of system-level frameworks in general, consider for example, a set of frameworks which could assist the developer in adapting new devices to the system. These frameworks could provide the foundation for supporting new and diverse devices such as networking and storage devices, as well as audio, video, MIDI, and animation devices.

In a traditional operating system, developers who need to support these new kinds of devices have to write entire device drivers for

each new device. However, with frameworks, developers only supply the characteristics and behavior specific to each new device. An immediate benefit to developers is that the generic code needed for each category of device is already provided by the framework, resulting in less code for the device driver developers to write, test, and debug.

Key Advantages of Frameworks

The overall benefit of frameworks is that they enable a higher level of code and design reuse than what is practical with other design approaches. In addition to frameworks, there are certainly many other reuse technologies such as 4GLs, code generators, and class libraries. However, 4GLs and code generators are based on procedural programming techniques and cannot easily provide the infrastructure and design guidance that are possible from frameworks. While class libraries do improve code reuse, they provide functionality at a very low level and force the developer to provide the interconnections between the libraries. These advantages of frameworks over procedural approaches and class libraries are much more difficult to create or recreate and constitute the real value of frameworks.

Also, the developer should realize that the benefits from frameworks and reuse are gained over time, since the productivity gains do not come just from the first or second use, but from multiple uses of the technology. We have touched on several benefits of using frameworks, but the following summarizes the major advantages:

- **Provide infrastructure and architectural guidance:** By virtue of the interconnections among the class libraries, much of the needed functionality already exists in the framework, thus reducing coding, testing, and debugging efforts. In addition, frameworks encourage better design in the code that developers do write by providing an "example" to guide them to more effectively utilize object technology. Applications developed with frameworks tend to be smaller, as well as more maintainable and reusable.
- **Provide a mechanism for reliably extending functionality:** While objects and object classes provide interfaces for extending functionality at a fine-grained level, frameworks provide this flexibility at a higher level. In this way, applications can be developed by using the framework as a starting point and writing smaller amounts of code to modify or extend the framework's behavior. These extensions can be added without sacrificing compatibility and interoperability because the interfaces are well defined.
- **Reduce maintenance:** Due to inheritance, when a framework bug is fixed or a new feature is added, the benefits of those changes become available more quickly to the derived classes. Also, changes are made only in one place, thus, the chance of introducing additional errors in the code is minimized.

Taligent's Approach To Frameworks

The complexity and lack of productivity in today's computing environments, as well as the stagnation in the computer industry, point directly to the need for a new software development approach based on frameworks. OOT can provide renewed vitality and productivity to the entire industry. However, in our view, the ultimate success of the emerging OOT paradigm hinges on the availability of object-oriented frameworks.

Up to this point, we have presented the software development problems faced by the industry and have shown how OOT and frameworks can address these problems and issues in new ways. Next, we will discuss how Taligent is extending the OOT and framework paradigms in order to create a new system software platform for innovation that greatly enhances a developer's ability to deliver more powerful solutions. Designed from the bottom up on OOT and frameworks, the Taligent environment will, as we've noted in the introduction of this primer:

- Empower developers to fully leverage OOT with frameworks that span the entire system. The pervasive use of frameworks in the Taligent system will deliver rich, built-in functionality at all levels and provide more computing value than when this functionality is added on as an option.
- Provide well-defined mechanisms that allow software and hardware developers to reuse our frameworks to extend and leverage this functionality for increased productivity and integration.
- Provide an integrated development environment designed for OOP that includes a wide variety of development tools all designed for rapid application development and customization.

This approach means that all the advantages of frameworks can be leveraged throughout the system: at the application level for such functions as graphical and text editing, data access, and user interfaces; and at lower levels of the system for services such as device drivers, printing, communications, file systems, and I/O.

Taligent Frameworks – An Architectural Perspective

Several application frameworks on the market today (such as the MacApp and OWL frameworks) take the architectural approach of layering objects and class libraries on top of procedural operating environments. While this approach has been a substantial improvement over procedural operating environments and has done a lot to introduce and promote OOT in the desktop computing industry, it creates its own unique limitations. Difficulties do not arise out of the use of layers, rather, they come from layers that are incomplete and because the layers do not fully leverage the benefits of OOT throughout lower parts of the system.

While object-layered systems have added value by pioneering object technology on top of procedural systems, we believe that the industry is ultimately moving towards systems that fully exploit OOT. Only a system built from the ground up with OOT can expect to achieve this goal. Taligent is taking this design philosophy in creating a system based entirely on OOT and frameworks.

Since fully object-oriented systems are optimized to take full advantage of OOT, software design is taken to a higher level of abstraction, allowing developers to more easily map the application to the business requirements of the problem. This enables development teams to maximize the use of their resources to produce applications more quickly and to innovate where it is currently impractical with today's systems.

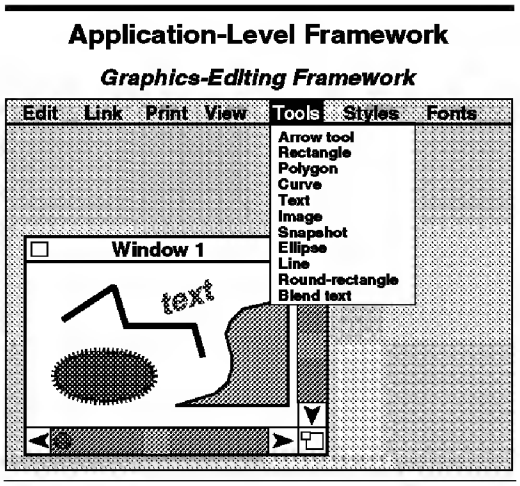
Only a system built from the ground up with OOT can expect to achieve full exploitation. Taligent is taking this design philosophy in creating a system based entirely on OOT and frameworks.

Taligent Application-Level Frameworks

Application frameworks in the Taligent environment will extend the entire scope and concept of frameworks that are available today to empower developers beyond the user interface. So, while the application frameworks available today are excellent examples of tools for developing applications, the greater value of frameworks is realized when they are applied in more pervasive ways at both the application and system levels. For example, at the application level, Taligent changes the whole concept of "applications." In the Taligent environment, programs are not embodied in old-style, monolithic applications; rather, users deal with information – the data that is critical to their needs and problem. There will be more information on this concept in later technology primers and white papers, but this is an indication of the scope of how Taligent is changing the current computing model.

Although it is premature to discuss the specifics of all Taligent frameworks, one example of an application-level framework that will be part of the Taligent environment is a graphics-editing framework **(see Figure 5).** This framework will provide all the common elements for 2-D and 3-D

Figure 5



graphics editing, such as support for the movement and layout of graphic objects, and tools for creating rectangles, polygons, and circles.

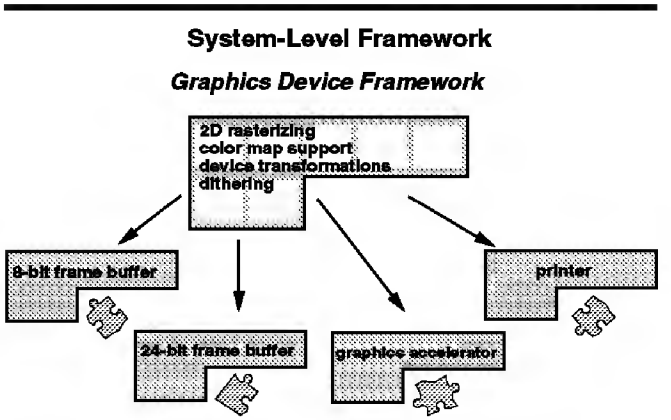
This type of framework could be used in two ways. First, consider a developer who is creating, for example, a fully integrated presentation package and needs graphics-editing capabilities (in addition to text editing, spell-checking, etc.). The developer could use the Taligent graphics-editing framework to enhance the presentation package with graphics-editing capabilities. This frees developers from having to build the graphics-editing software themselves and allows them to concentrate on areas where they can add the most value. A second use would be to support incremental development of more sophisticated graphics editors or graphics editors for specific markets. Since the framework provides most of the common elements of graphics editing, a developer could start with the Taligent framework and add functionality for their particular market, thus reducing their start-up costs.

Taligent System-Level Frameworks

To envision the application of Taligent frameworks at the system level, suppose the developer has a graphics device framework as depicted in **Figure 6.** The framework could provide preexisting functionality, such as 2-D rastering, color map support, device transformations, and dithering for any device from "dumb" frame buffers to graphics accelerators. This type of framework could allow easier additions of new graphics devices since the developer only needs to code what is different about each device.

To support an 8-bit frame buffer, for example, a developer would inherit most of the preexisting functionality from the framework, so the amount of code the developer writes would be relatively small. However, if an OEM is supplying a board for a 24-bit frame buffer, the OEM does not need color map support or dithering. So, in this case, the OEM developer either does not inherit those methods or does not use them to implement the 24-bit driver. There might be some additional features that developers want to add, so, the puzzle piece might be a little larger than with the 8-bit frame buffer.

Figure 6



Now, suppose a system vendor needs to create a driver for a graphics accelerator. In this case, the driver writer starts with the framework and subclasses and replaces code that has been written to do the 2-D rastering (how to draw a line or a fill or how to do a transfer mode), and replaces it with a direct call to the hardware capability that is in the board. In this manner, the developer can choose not to use the code in the Taligent system and instead replace it with code that uses the hardware to implement these methods.

To create raster printer drivers, developers could again subclass from the framework, and implement/override the default methods to support a wide range of printer functionality.

Because these new devices are added to existing system frameworks, all applications can take advantage of new system extensions and functionality without modification. Since the applications access functionality at a higher abstraction, they get the powerful, new implementation for "free."

Taligent Development Environment Frameworks

In addition to providing frameworks at the application and system levels, Taligent believes it is just as important to apply the concept of frameworks to the development environment in order to increase developer productivity. To do this, we will offer powerful and customizable development tools for developing, implementing, and debugging object-oriented software. The development environment is designed to speed program development by supporting an incremental and interactive approach to writing, debugging, and testing programs. In this environment, programmers can focus on solving one problem at a time, rather than solving many problems at once.

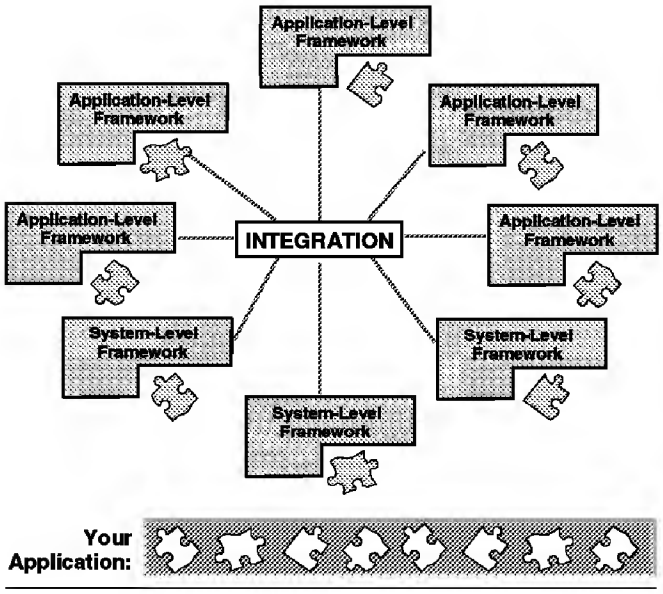
Like other parts of the Taligent environment, the development system is built on the concept of frameworks. So, over time, as the development environment evolves, developers will be able to subclass from the development environment frameworks and inherit from the existing functionality to create their own custom tools such as viewers, browsers, editors, debuggers, compilers, and on-line documentation.

How Taligent is Changing the Programming Model

Since the Taligent environment is based on framework-oriented technology, developing software in this environment consists of writing the domain-specific puzzle pieces that adhere to the protocols of the various frameworks **(see Figure 7).** Through framework-oriented technology and programming, the Taligent environment endeavors to change the whole concept of programming and dramatically increase developer productivity. Programming in the Taligent environment is accomplished by deriving classes from the extensive suite of preexisting frameworks, and then adding new behavior or overriding inherited behavior as required.

Figure 7

Taligent Environment Programming



The developer's application becomes the collection of code that is written and shared by other framework programs. This is a powerful concept because it allows developers to concentrate on what is essential and different about each particular problem and write only the code needed by the frameworks. And, since developers start from the same base, they are able to build on each other's work. In this world, overhead and error rates are greatly reduced, and software simply gets completed and delivered more quickly.

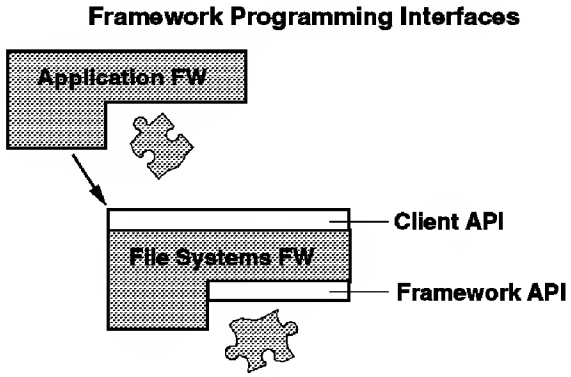
This powerful model is possible because of the framework's programming interfaces. Often, frameworks are represented at the top of the system hierarchy with one set of application programming interfaces (API). However, frameworks, particularly as they are applied at the system level, actually have two APIs operating, as shown in **Figure 8.**

The first API is the client API which is used by other frameworks or application developers. This API is for developers who just want a simple abstraction and to simply be a user of the framework services. For example, as illustrated in Figure 8, an application

framework that needs to write out data to a file, would call the client API of the file system framework.

The second API is known as the framework API and is for developers who want to take advantage of the flexibility and extensibility that frameworks provide. This API provides all the hooks to the developer's code (the puzzle pieces) that modify the framework's behavior to provide the desired software- or hardware-based solution. It is through this interface that the principle of "don't call us, we'll call you" is implemented. In some cases, the amount of developer code is minimal. In other cases, the developer might make extensive modifications and create something completely new.

Figure 8



Domain-Specific Frameworks

By providing a framework-based system, the Taligent environment presents an ideal platform for creating and deploying innovative domain-specific solutions. It is common for developers to write a family of closely related programs. For example, an independent software vendor might be producing a product line for learning foreign languages. Or a corporate developer might need to provide custom accounting programs for different subsidiaries in the organization. In either case, the job can be made easier by capturing common code in a framework.

Traditionally, this problem might have been attacked by providing domain-specific procedural libraries of functions. Or, better yet, the code might have been combined into class libraries of what were hoped to be reusable objects. Unfortunately, this has often not proved to be successful because it is hard to provide default behavior in a library. Using a library typically involves reading the interface to determine which functions are available to call and then trying to figure out which ones are needed to be called and in which order. It is usually so difficult to figure out how to use and customize the library that the developer ends up rewriting most code from scratch for each new project.

Frameworks are a better way to embody domain expertise because the developer can provide default behavior in the framework. Therefore, the user of the framework does not have to know how or when to call each function – the framework is already wired to call the right things at the right time. All the developer needs to know is where the hooks are to specify and extend the framework's behavior.

Summary

Improving developer productivity is a major challenge for the entire industry. While current approaches have advanced productivity, the next generation of software must fully exploit OOT in order to provide the productivity and development leverage that is needed to solve today's complex computing problems.

While the use of object technology has demonstrated the capability to significantly increase developer productivity and enhance program maintainability, Taligent contends that is not just a matter of switching to OOT, but how this technology is implemented. The success of OOT hinges on an infrastructure like frameworks that enables developers to change their programming mindset, design software that is more reusable and maintainable, and create innovative software that addresses business problems.

Frameworks and systems that are based on frameworks, such as the Taligent environment, empower developers to fully realize the potential of improved design and code reuse, including reduced development requirements, reduced maintenance, and higher reliability. In addition, programming based on frameworks enable the following changes:

- **Developers can focus on their true "value-added":** Just as standard programming interfaces insulate software routines from system dependencies and standard utilities facilitate development, frameworks enable software developers to concentrate on application solutions and rely on the framework to provide consistent services. This frees developers who are not necessarily experts in a certain area from the complexity of the underlying details. In this manner, frameworks encourage an environment of solving domain problems instead of programming problems.
- **Good design practices and proliferation of expertise:** Good software design in a particular area requires domain knowledge that is typically acquired only by experience. Corporate and commercial development organizations, and systems integrators as well, have this acquired experience in particular areas, such as manufacturing, accounting, insurance, or financial instruments. Frameworks allow organizations to package the common characteristics of that expertise by embodying it in the organization's code. Frameworks and the embodied expertise behind the frameworks have a strategic asset implication for internal use and for those organizations who see business opportunities for reselling specialized knowledge. For example, frameworks give systems integration companies with expertise in vertical markets a distribution mechanism for packaging, reselling, and deploying their expertise.
- **Improved consistency and compatibility:** Because frameworks embody expertise, problems are solved once and the business rules and design are used consistently. This allows an organization to build from a base that has been proven to work in the past. Another advantage is that since different applications share the same "DNA" from the system, the applications can work together (for example, cut-copy-paste or drag-and-drop) in more substantial ways. The resulting system is better integrated from a user's point of view, while requiring less work by developers to get their programs to work together.

Taligent wants to shift the process of development away from today's focus on working around operating system barriers. Simplifying development efforts with frameworks and an object-oriented development environment allows corporate and commercial software developers, as well as hardware OEMs, to focus on innovations that address real-world business problems and issues.

Frameworks also create the potential for a new business model where the pace at which new ideas are delivered to the marketplace is dramatically accelerated and the cost of entry for a new product is limited only by the value of the idea. The renewal of innovation in the industry will restore vitality and fuel new growth opportunities for the entire computing industry.

As a result, a whole new marketplace is enabled where every player in the computing industry stands to gain tremendous advantages. Commercial software developers, value-added resellers, and system integrators can gain increased productivity through design reuse, more manageable development, and an improved ability to deal with complexity in data, networks, and logic. Corporate developers attain a better linkage between business needs and the supporting applications. This empowers software to work as a catalyst, enabling companies to rapidly exploit new opportunities, rather than acting as a bottleneck.

Frameworks provide system OEMs the ability to differentiate themselves economically by allowing an easier way to package new functionality with systems. And finally, end-users will reap the advantages of not only seeing today's horizontal software produced faster and more economically, but of a whole new category of smaller, more innovative tools that are more tailorable to their unique needs.

Next Steps

We hope this primer has provided some insight into frameworks and how you can benefit from them. In order to learn more about OOT and frameworks and how to prepare for the Taligent environment, we encourage developers to:

- **Learn OOP:** Any object-oriented language can help you learn OOP. C++ is the first language that will be supported in the Taligent environment, but it is expected that other OO languages such as Smalltalk will be supported.
- **Learn object-oriented design (OOD):** Developers who simply use C++ as a better C without fundamentally changing their design approach will not realize the real benefits of object technology. Also, doing a good job of OOD requires more than just learning an OOP language. The whole point of OOP languages is to allow a different approach to software design. This approach takes time and energy to learn. A good method of learning OOD is to study the design of mature frameworks and learn by example. Also, there are several books and commercially available training courses to help learn OOD.
- **Learn to design and create your own domain-specific frameworks:** Begin with class libraries and understand both their power and limitations. Then begin to think in terms of frameworks and framework-oriented programming, both at the application level and the system level. Becoming familiar with commercially available frameworks will also assist in the learning process. Even though Taligent will provide a wide range of frameworks, remember that the even greater value in frameworks is realized when you create your own for your particular problem domain.
- **Partner with Taligent:** You are invited to come to us with questions and ideas. We especially encourage corporate and commercial developers who have new ideas for applications that are not possible or feasible on today's systems. You can also request to be placed on the mailing list for future business and technology primers and white papers. Contact us by sending e-mail to *developer_partnership@taligent.com* or calling 408-862-7465 (408-TO-B-PINK).

Additional Reading

For additional information on object technology and frameworks, we suggest the following sources:

The C++ Programming Language, Second Edition, Stroustrup, Addison-Wesley

Developing Object-Oriented Software for the Macintosh, Goldstein and Alger, Addison-Wesley

Object-Oriented Design with Applications, Booch, Benjamin/Cummings

Object-Oriented Technology: A Manager's Guide, Taylor, Addison-Wesley

Taligent White Papers

Below is a list of available Taligent white papers and backgrounders:

A Study of America's Top Corporate Innovators, Taligent, Inc., 1992

Lessons Learned from Early Adopters of Object Technology, Taligent, Inc., 1993

Driving Innovation with Technology: Intelligent Use of Objects, Taligent, Inc., 1993

10725 N. De Anza Boulevard
Cupertino, CA 95014-2000
408 255 2525

Produced in the USA.

TT2/6/93